CS 181 Notes

Franklyn Wang

April 29, 2019

Contents

§1 January 28th, 2019

§1.1 Overview

Machine learning is changing the world (think about everything). However, it can do a lot of evil things (think about bad tweets). The steps to machine learning.

1. Make appropriate model choices
2. Have sufficient understanding to learn and apply the new techniques
3. Identify sources of error
4. Evaluate carefully

§1.2 What is Machine Learning?

There is a task or a goal. We have past data (experience). We have a performance measure. We have data now, and we have a prediction (action) now.

This starts with linear regression, which people have done for 95 years. Modern day machine learning is different from linear regression not just that they are big ("big data") but also from complex structures and complex sources. Part of the zen of machine learning is determining what the input and output should be.

§1.3 A Story: Treating Depression

One question that people think about is, which meds do we give to patients who are depressed?

The goal is to treat major depressive disorder. We have past patient data, the output is whether the drug is successful. The specific patient is the input and the drug suggestion is the output.

Large-scale clinical trial decided between certain features or pairs of drugs. MRIs and biomakers can be used to determine subtype and treatment choice. We want to be able to recommend all common drugs, for patients with any possible prior treatment history.

We have a site A and a site B, which have different cohorts of patients. This is a hard problem, since a drug succeed criteria is hard.

If the doctor keeps prescribing a drug, that must mean that something is going right.

In high dimensions, there is a tension between sparsity and interpretability. For example, "pregnant" could code for "female", and "hip fracture" could code for "elderly". We can use topic models, which group codes together which are correlated.

This method actually scores well in interpretation, since many of these things can be interpreted.

§1.4 Detecting Fake Videos

How do we differentiate deepfakes from real videos? Someone suggests that

§1.5 Machine Learning Taxonomy

Machine learning breaks into lots of categories, like supervised learning, like classification, when we're finding a discrete type, like Swype phone letters. In regression, you want to output an outcome, which is continuous. Then there is something called unsupervised learning. There is a task, and a data. Discrete unsupervised learning is something called a classification, like news. Another form is an embedding, when you map them onto a continuous object, like point onto a line.

And lastly, there's something called reinforcement learning, where we have histories.

§1.6 Structure of the Course

We can start with basic regression problems. We will then discuss model selection and evaluation. We will then expand the cube, and talk about regression and classification, as well as clustering vs embeddings, as well as reinforcement learning.

§1.7 Nonparametric Regression

In regression, we have continuous inputs and continuous outputs. There are a few bad solutions to this problem.

1) *K-nearest neightbor*

   1. Find the K nearest points to x^*.
   2. Find the mean of these points.

   The advantage of this method is that it is very flexible. The cons are that we have to choose the meaning of the word "nearest", and that this can be complicated, as we need a metric. Furthermore, you have to keep all the training data.

2) *Kenerlized Regression*

   We predict
$$\hat{y}^* = \frac{\sum_n K(x^*, x_n) y_n}{\sum_n K(x^*, x_n)}$$
   where K represents a "similarity function". This makes sense intuitively, as K measures how close two points can be.

§1.8 Conceptual Exercise

Assume that we choose a specific form $K(x', x) = \exp\{-(x - x')W(x - x')^\top\}$.

a) Will the predictor $\hat{y}^*$ ever be $< \min(y_n)$ or $> \max(y_n)$?

§2 January 30th, 2019

§2.1 Non-parametric Regression

Last time: we went over non-parametric regression. Non-parametric regression has infinitely many parameters. This is a regression that "grows" with the data. This is either a feature or a bug. It's possible that the regression gets more sophisticated with time, but a bug is that it doesn't become substantially similar.

One of these approaches is K-nearest neighbors, where you take the K nearest points to x^*, and then average their values. One subtle variant is to find all training points within ϵ of x^*.

The last thing we talked about was a kernelized regression. We have the predictor that looks like
$$\hat{y}^* = \sum_n \frac{K(x^*, x^n)}{\sum_{n'} K(x^*, x^{n'})} y_n$$

One example of a kernel function is

$$K(x, x') = \exp\{-(x - x')^\top W(x - x')\}$$

So now we have a question of what W should be. It is not entirely clear what it should be. Therefore, we need to *train* W. We want to minimize some objective function, say, loss.

Choose the loss function as

$$\mathcal{L}(W) = \sum_n (y_n - \hat{y}_n)^2$$

where $\hat{y}_n$ are the predictions for the values. To minimize a squared loss function, the answer is just to let W be ∞. If you only look at things that are close to yourself, you will be *very* accurate. One way you can change this is to let the estimator be take-one-out, in other words, take the sum only over elements that are not exactly equal.

§2.2 Linear Regression

Now we choose a parametric form for $\hat{y} = f(x)$, and seek to minimize a loss. For linear regression, the form is $\hat{y} = w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_D x_D$.

Note: Here, $\langle x_1, x_2, \ldots x_D \rangle$ forms a single data point!

Now, we make a small notation change: let

$$\mathbf{x} = [1\, x_1\, x_2 \ldots\, x_D]$$

Now we can write

$$\hat{y} = w_0 x_0 + w_1 x_2 + \ldots + w_D x_D = \mathbf{w}^\top \mathbf{x}$$

It make the rest of the notation somewhat easier.

The general formula here is that first we choose a regressor, then we choose a loss, and finally we optimize regressor for the loss.

Let our loss be the least squares loss function. Let

$$\mathcal{L}(\mathbf{w}) = \frac{1}{2}\sum_n (y_n - \hat{y}_n)^2 = \frac{1}{2}\sum_n (y_n - \mathbf{w}^\top \mathbf{x}_n)^2$$

Option A is to take some gradients. We compute

$$\nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}) = \sum_n \underbrace{(y_n - \mathbf{w}^\top \mathbf{x}_n)}_{\text{scalar}}(-1)(\mathbf{x}_n)$$

Taking derivatives with respect to vectors is useful. This information is in the *matrix cookbook*. Now we perform gradient descent. Gradient descent is simply applying the update rule:

$$\mathbf{w} := \mathbf{w} - \eta \nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w})$$

From a programmer perspective, gradient descent is very cheap in terms of programmer time, so in cases where there is an analytic solution, those are desired.

Option B is to solve this analytically. We can get

$$\mathcal{L}(\mathbf{w}) = \frac{1}{2}(\mathbf{y} - \mathbf{w}^\top \mathbf{X})(\mathbf{y} - \mathbf{w}^\top \mathbf{X})^\top = \frac{1}{2}(\mathbf{y}\mathbf{y}^\top - 2\mathbf{w}^\top \mathbf{X}\mathbf{y}^\top + \mathbf{w}^\top \mathbf{X}\mathbf{X}^\top \mathbf{w})$$

Computing

$$\nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}) = -\mathbf{X}\mathbf{y}^\top + \mathbf{X}\mathbf{X}^\top \mathbf{w} = 0,$$

we get the solution

$$\mathbf{w}^* = (\mathbf{X}\mathbf{X}^\top)^{-1}\mathbf{X}\mathbf{y}^\top$$

We can interpret this statistically. $\mathbf{X}\mathbf{y}^\top$ represents the correlation between $\mathbf{X}$ and $\mathbf{y}$. It's larger if the variables are correlated. With this interpretation, $\mathbf{X}\mathbf{X}^\top$ is essentially a normalizer.

The linear algebraic interpretation is that we want something such that

$$\mathbf{w}^\top \begin{bmatrix} | & | & \cdots & | \\ \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_N \\ | & | & \cdots & | \end{bmatrix} = [y_1 \cdots y_N]$$

Instead, this can be thought of as a projection.

§2.3 Conceptual Questions

If there are outliers, how does this affect linear regression? How do you detect non-linearity. To detect outliers, you can exclude a data point, and that will tell you the result.

§3 February 4th, 2019

When the guy messages first, the median age looks like this. 20 year old men message 22 year old women, 45 year old men go after 39 year old women.

The point of these pictures s that there are different stories.

We also go into discussion about trade off between size of reproductive organs as well as brain size. In bats, there are tradeoffs.

"It's Harvard. Grades are inflated."

§3.1 Regularization

Assume that you get something like

$$y = w_0 + w_1 \begin{pmatrix} \text{your} \\ \text{height} \\ \text{in} \\ \text{inches} \end{pmatrix} + w_2 \begin{pmatrix} \text{your} \\ \text{length of leg} \\ \text{in} \\ \text{inches} \end{pmatrix}$$

after running a linear regression. If you got $w_2 < 0$, what would you think? The way something like this happens is that the two variables are correlated.

Recall last time we had a calculus, a statistical, and linear algebraic view of optimization / solution. We had a predictor

$$y = w_0 + w_1 x_1 + \dots$$

The least-squared loss is

$$\frac{1}{2}\sum_n (y_n - \hat{y}_n)^2.$$

The linear algebra view is that least-squared loss corresponds to doing the orthogonal projection of Y into X space.

§3.2 Likelihood

We now look at a generative model, or a probabilistic view. Let x_n be taken from some dataset, or distribution. Then we create $f_n = w^\top x$. We will take $\epsilon_n = \mathcal{N}(0, \sigma_n^2)$. Then $y_n = f_n + \epsilon_n$.

We now want to evaluate

$$\mathbb{P}(\text{data}|\text{model}).$$

Theorem 3.1 (Gaussian PDF)
We have that

$$\mathcal{N}(z; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(z-\mu)^2\right).$$

Now we compute

$$\mathbb{P}(Y|X, w, \sigma^2) = \prod_{n=1}^{N} p(y_n|x_n, w, \sigma^2)$$

This is terrifying on account of the *product* so we take logarithms.

Now, maximizing log probability is simply just maximizing:

$$\sum_n \log p(y_n|x_n, w, \sigma^2).$$

We get that this is equivalent to

$$\sum_n \log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) - \frac{1}{2\sigma^2}(y_n - w^\top x_n)^2$$

where the least-squared loss function appears. This gives us the key observation - least-squared loss corresponds to Gaussian Noise.

Now, we solve for σ. Taking derivatives with respect to σ^2 gives us

$$0 = \frac{-N}{2} \cdot \frac{1}{\sigma^2} - \frac{1}{2}(y - w^\top X)(y - w^\top X)^\top \left(\frac{1}{\sigma^2}\right)^2.$$

Now we add to the story. Let $w \sim p(w)$. Then for x_n in $\{1, \ldots, N\}$, the goal is now to optimize the joint probability of $w_n\{y_n\}$. This part is

$$\prod_{n=1}^{N} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{-\frac{1}{2\sigma^2}(y_n - w^\top x_n)^2\right\}$$

One example of a prior on w is $\mathcal{N}(0, \mathbf{I}\sigma_n^2)$. We can find a joint PDF for w as

$$\prod_{d=1}^{D} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{-\frac{1}{2\sigma_w^2 w_d^2}\right\}$$

§4 February 6th, 2019

Say that we can modeling mortgages. Assume that you have a few people who might default on a loan.

The model was that of convenience and made an overly strong hypothesis. If you just stick in a Gaussian, you can create the 2008 financial crisis.

From last class, we have that

$$\hat{y} = w^\top x + \epsilon.$$

Example 4.1 (Concept Question)
The standard linear model for $y = ax + \epsilon$ has ϵ modeled as a Gaussian. But what if we let ϵ be a Laplace distribution, or one with PDF $(1+\epsilon^2)^{-1}$?

The Laplace distribution minimizes L_1 loss. This is "robust" to noise, since it's not easily perturbed by outliers.

T-distributions are also robust, because the heavy tails make them okay with large errors.

§4.1 Model Selection

Example 4.2 (Bias-Variance Tradeoff)
The goal of model selection is to find a model that generalizes to new x. If we fit a model and it's systematically wrong, this is called **bias**. If you make a model and it's too complicated, there's variance.

Example 4.3 (Curse of Dimensionality)
Let X have 8 data points with $D = 2000$, such that the variables are binary. Let y equal $x_{d=1}$. What's the probability that another dimension d also matches y, totally by accident? The answer is not hard to find: it's

$$1 - \left(\frac{255}{256}\right)^{D-1} \approx 1.$$

§4.2 Validation

Given a dataset, we split it into training, validation and a test group. We should train all models on the training data, and then evaluate the models on the validation data. We will pick the best and report it's score on test.

One thing to deal with this is to use cross-validation. Here, you break the data into K chunks. You train on $K-1$ chunks and then test on a chunk.

§4.3 The BIG DATA revolution

Here we show why big data is really useful. We're trying to minimize

$$E_y[(y-\hat{y})^2] = E_y[(y - f_D(x))^2]$$

f_D was found with some given dataset.

Now we can write this as

$$E[(y - \bar{y} + \bar{y} - f_D(x))^2] = \underbrace{E[(y-\bar{y})^2]}_{\text{noise}} + \underbrace{E[(\bar{y} - f_D(x))^2]}_{\text{model error}}.$$

What we can control is $\mathbb{E}[(\bar{y} - f_D(x))^2]$. Now, this expression is

$$E[(\bar{y} - \bar{f}(x) + \bar{f}(x) - f_D(x))^2] = \underbrace{\mathbb{E}[(\bar{y} - \bar{f}(x))^2]}_{\text{bias}} + \underbrace{\mathbb{E}[(\bar{f}(x) - f_D(x))^2]}_{\text{model variance}}$$

High amounts of data can make model variance lower, showing the power of big data.

§5 February 11th, 2019

§5.1 Frequentist Model Selection

There was a Kaggle contest with prostate cancer. Someone added a feature which measured whether the patient's treatment had started - suddenly the challenge became very easy.

There's something called causality leakage. If the clinician suspects something, orders tests. The AI notices that there are tests and create an alert. Then the event happens, and the AI thinks it succeeded.

Another Kaggle contest was created to look for breast cancer.

The current setting is that we have (x, y) pairs and that they are IID. Last time:

$$\mathbb{E}[(y-\hat{y})^2] = \text{noise in data} + \text{bias in model}^2 + \text{variance in model}$$

High variance corresponds to overfitting.

Another problem is underfitting because the model did not try very hard.

After/As you measure these kinds of errors, how do you address it?

1. Regularization: Here, we try to minimize an expression of the form
$$\min_w \mathcal{L}_{\mathcal{D}}(w) + \lambda R(w)$$
   - L^2 regression forces coefficients to be small, but not necessarily zero.
   - L^1 regression wants to force coefficients to go to zero.
   - The horseshoe loss is steep near zero, but less steep as things go out.
2. Ensembles: train many predictors and average. Some examples include random forest, bagging, boosting / gradient boosting. Ensemble methods control variance while keeping bias the same.

We have two models.

	Model A	Model B
Train	0.95	0.72
Test	0.6	0.7

Model A is overfitting, since it does really well on the training data. Model B is unlikely to become substantially better with extra training data. Model A will get a different answer on a new dataset, but model B is likely to become more stable.

§5.2 Bayesian View of Model Selection

The Bayesian machine learning model focuses on marginal likelihood. It's noted that this is fairly challenging. If you can cross-validate, and your have a large amount of data, Bayesian persepctive is very strong.

You might put priors over parameters. Therefore, x_n produces y_n, and σ^2 was affecting y_n. Finally, σ_w^2 was producing w. Frequentists view that parameters are fixed, and that data is random. In the Bayesian paradigm, the parameters are random. We have a $p(w)$, but the data is fixed. Now we can compute $p(w|\mathbf{X}, \mathbf{Y})$. By Bayes' rule, we have that

$$p(w|X, Y) \propto p(Y|X, w)p(w)$$

Then we want to find $p(y^*|x^*, X, Y)$, and how do we find it? An integral. We now take

$$p(y^*|x^*, \mathbf{X}, \mathbf{y}) = \int_w p(y^*|x^*, w)p(w|X, Y).$$

Then we get

$$\mathbb{P}(Y|X) = \int p(Y|X, w)p(w)$$

Example 5.1 (Coins)
We have coins that come up as "1" with probability θ. We can write that

$$p(x|\theta) = \theta^x(1-\theta)^{1-x}$$

Now we want to maximize

$$p(x_1, \dots x_n|\theta) = \prod_n \theta^{x_n}(1-\theta)^{1-x_n} = \theta^{N_1}(1-\theta)^{N_0}$$

Now let's put a prior on θ. Now recall from statistics that the beta distribution is proportional to $\text{Beta}(\theta) \propto \theta^{\alpha-1}(1-\theta)^{\beta-1}$ We now get that

$$\mathbb{P}(\theta|X) \propto \mathbb{P}(X|\theta)p(\theta) = \theta^{N_1+\alpha-1}(1-\theta)^{N_0+\beta-1}$$

A prior of $\text{Beta}(\alpha, \beta)$ is updated to $\text{Beta}(\alpha + N_1, \beta + N_0)$.

If your prior is that $\alpha = 2, \beta = 2$. There is not truth in the prior. Priors arise from prior experience. If there are two heads, likelihood is θ^2, so $\theta = 0$ maximizes it. Now we have that $\mathbb{P}(\theta|\text{data}) = \theta^3(1-\theta)$. We get an $\alpha' = 4$ and a $\beta' = 2$.

Maybe if you get two heads, you shouldn't be immediately certain that the coin flip will always be heads. We find

$$p(x^*|x_1, x_2) = \int_\theta \theta p(\theta|x_1, x_2)d\theta = \mathbb{E}[\theta] = \frac{\alpha + N_1}{\alpha + \beta + N_1 + N_0}$$

The mode of a distribution and the maximum likelihood estimate are different. The posterior predictive is what a true Bayesian would do.

§6 February 13th, 2019

There was a tool for automatic tumor classification for tomography. Basically, many people began using this system. Post-deployment surveillance is really important. It turns out it was doing poorly, because in clinical trials doctors were trying harder than they were in real life.

Example 6.1 (Concept Question)
Assume our data is the points $(0,0)$, $(1,1)$. Let $y = a_0 + a_1 x + a_2 x^2$.

1. What can we say about a_0, a_1, a_2?
2. If $p(a_0) = \text{Unif}[-100, 100]$, $p(a_1) = \text{Unif}[-1, 1]$, $p(a_2) = \text{Unif}[-100, 100]$, then what is your posterior predictive at $x = 1/2$.
3. What is the posterior predictive if the model is linear?

We have that $a_0 = 0$, and $a_1 = 1 - a_2$. $a_1 = \text{Unif}[-1, 1]$, $a_2 = 1 - a_1$, $a_1 * 1/2 + (1 - a_1) * 1/4 = 1/4 + 1/4a_1$. The posterior predictive is $\text{Unif}[0, 1/2]$. The predictive posterior if the model was linear is $1/2$.

§6.1 Classification

In classification, we have continuous inputs and discrete outputs.

In linear classification, we have that

$$\hat{y} = \text{sgn}(\underbrace{w_0}_{\text{offset}} + \underbrace{w^\top x}_{\text{direction}})$$

One loss function is 1 if $z > 0$ and 0 otherwise. For example, $\ell_{0,1}(-y \cdot (w_0 + w^\top x))$. Now, we're going to use an alternative loss function, called **hinge loss**.

Let

$$\ell_{\text{hinge}}(z) = \begin{cases} z \text{ if } z > 0 \\ 0 \text{ else} \end{cases}$$

We're going to give partial credit if it's relatively close to the boundary (many small errors is worse than one large error). The convenient thing here is that derivatives are usually actually defined. Now, we have a derivative

$$\frac{\partial \mathcal{L}}{\partial w} = -\sum x_n y_n.$$

The thing with gradient descent is that if you compute it on fewer data points it will move faster. This is called *stochastic gradient descent.*

§6.2 Metrics

If $y = \hat{y} = 1$, this is a true positive. If $y = 0, \hat{y} = 1$, there is a false positive. If $y = 1, \hat{y} = 0$, this is a false negative. If $y = 0, \hat{y} = 0$, this is a true negative.

We can look at the number of true positives over the number of true and false positives. This refers to **precision**.

§7 February 20th, 2019

Last time we were talking about metrics for classification. Remember, precision is

$$\frac{TP}{TP + FP}.$$

The recall is equal to

$$\frac{TP}{TP + FN}$$

This looks at all the cases where there was a real problem, what was happening. Recall is important when you don't want to miss any alarm situations. The false positive rate is

$$\frac{FP}{FP+TN}$$

The ROC is the false positive rate vs the true positive rate. You can always get (x, x) by making something that sets an alarm exactly x time.

One metric for the quality of an ROC is simply the area under the curve.

Concept question: Suppose that you have a classifier with an AUC $= 1/2 + 1/8$. What are the possible true positive rates if a false positive rate is equal to $1/4$? The answer is $1/4, 3/4$. A drawing actually helps substantially here.

ROC curves are convex for the following reason: you can always get any convex combination.

§7.1 Discriminative Classifier

We're trying to create a model to compute

$$\mathbb{P}(y = c_k|x)$$

In the binary case, we can use $\sigma(z) = \frac{1}{1+\exp(-z)}$. For linear probabilistic classification, we have

$$\mathbb{P}(y = 1|x) = \sigma(w_0 + w^\top x)$$
$$\mathbb{P}(y = 0|x) = \sigma(-(w_0 + w^\top x))$$

Now the loss function over N examples is equal to

$$\mathcal{L}(w, w_0) = \sum_{n=1}^{N} y_n \log \hat{p}(y_n = 1|x_n) + (1 - y_n) \log \hat{p}(y_n = 0|x_n)$$

The generative model is a generative model, y, which generates something x. This model is more complex, but missing data is not an issue. Now we can use Bayes' rule, which states that

$$\mathbb{P}(y = c_k|x) \propto \mathbb{P}(x|y = c_k)\mathbb{P}(y = c_k)$$

Now suppose that y causes x_1, x_2. Then we have that

$$\mathbb{P}(x_1, x_2|y) = \mathbb{P}(x_1|y)\mathbb{P}(x_2|y)$$

The upshot of all of this is that this model can incorporate multiple pieces of information.

If X is modeled as a Gaussian, we have that

$$X \sim \mathcal{N}(\mu_1, \Sigma_1)$$

if $y = c_1$ and

$$X \sim \mathcal{N}(\mu_2, \Sigma_2)$$

if $y = c_2$.

If $\mathbb{P}(y = c_1|X) > \mathbb{P}(y = c_2|X)$, we will guess c_1, and guess c_2 otherwise.

Now, we're going to compute

$$\log(\text{terms not depending on } x) - \frac{1}{2}\left[x(\Sigma_1^{-1} - \Sigma_2^{-1})x^\top - 2x(\Sigma_1^{-1}\mu_1^\top \mu_2^\top)\right]$$

If covariances are equal, the decision boundary is linear. Otherwise, the decision boundary is quadratic.

§8 February 25th, 2019

Neural Networks are considered useful.

§8.1 Naive Bayes

Naive Bayes simply says that

$$\begin{aligned}\mathbb{P}(y|x) &\propto \mathbb{P}(x|y)p(y)\\ &= \mathbb{P}(x_1|y)\mathbb{P}(x_2|y)\mathbb{P}(x_3|y)\mathbb{P}(y)\end{aligned}$$

Note that this depends on conditional independence of the x_1, x_2, x_3 given y.

§8.2 Algebra

Suppose that we have $w^\top x + w_0 = \log(p(y=1|x)) - \log(p(y=0|x))$. Now, substitute $p(y=0|x) = 1 - p(y=1|x)$. We can now write $w^\top x + w_0 = \log(p(y=1|x)) - \log(1-p(y=1|x))$. This implies that

$$\mathbb{P}(y=1|x) = \frac{1}{1+\exp(-1(w^\top x + w_0))}$$

This is the essence of logistic regression.

Now we can find a likelihood. We can find

$$-\sum_n y_n \log(1+\exp(-f_n)) + (1-y_n)\log(1+\exp(f_n))$$

After this we can apply a softmax function, which gives us

$$\frac{\exp(f^1)}{\exp(f^1)+\exp(f^2)+\ldots+\exp(f^k)}.$$

We can then look at log softmax instead of log-likelihood.

Of course, taking DERIVATIVES is important. We obtain

$$\frac{\partial \mathcal{L}_n}{\partial w} = \frac{\partial \mathcal{L}}{\partial f_n}\frac{\partial f_n}{\partial w}$$

Now, let $f = w^\top \phi + w_0$. Assume that $\phi_j = \sigma(w_j^\top x + w_{j0})$.

§8.3 What can Neural Networks Learn?

One insight that we get is that we can learn an XOR with a neural network. For example, if $x = [0,1]$ or $[1,0]$ we get $y = 1$, and if $x = [0,0]$ or $x = [1,1]$ we get $y = 0$. We now can take $\phi_1 = \sigma(-x_1 - x_2 + 1/2)$. This picks out the point $(0,0)$ particularly strongly. Then we can take $\phi_2 = \sigma(x_1 + x_2 - 1.5)$. This will pick out $[1,1]$.

Another thing that neural networks can learn is a circle pattern. Note that if we have a ton of lines, we can basically approximate what a circle looks like.

Now, we can learn what convolution neural networks are. If you image an image, and you take small patches with filters, and then do pooling, and so on.

§8.4 Concept Question

Problem 8.1. When can unlabelled data help?

Probabilistic classification works this ways. There's a hidden truth, and the output may or may not be Noisy. Let y come from a Bernoulli. Then let x come from some MVN if $y = 0$ and some other MVN if $y = 1$.

§9 February 27th, 2019

It turns out that neural networks can be used for things besides for making money, like doing science. When doing finite element analysis, they used a neural network to greatly speed up calculations. There are situations in which data is cheap.

When does unlabeled data help? It can help you, say, figure out how close you are to a decision boundary.

§9.1 Neural Network Architectures

You take a vector x, do a filter (which makes the size stay the same), then do a pooling, then another filter, and then more things. The advantage is that a filter can look for a pattern anywhere. There are also recurrent neural networks, where you need to keep track of words that were used in the past to predict words in the future. There are many types of these.

Now there's the autoencoder, where we want to capture the "essence" of things.

§9.2 Neural Network Optimization

We will now derive back-propogation. We can define

$$\mathcal{L}_n(\{W\}) = -\frac{1}{2}(y_n - f_n)^2,$$

where W represents all the parameters in the system. One can show that

$$\frac{\partial \mathcal{L}_n}{\partial W} = (y_n - f_n)\frac{\partial f_n}{\partial W}$$

For classification, we can use

$$\mathbb{P}(y_n = 1 | x_a) = \frac{1}{1 + \exp\{-f_n\}}.$$

Then we have that

$$\mathcal{L}_n(W) = y_n \log p_n + (1 - y_n)\log(1 - p_n).$$

The important idea here when doing the differentiation is the vector valued chain rule. If we have that $y = f(g(h(\mathbf{x})))$ If these functions were all scalar functions, then $\frac{\partial f}{\partial x} = \frac{\partial f}{\partial u}\frac{\partial u}{\partial v}\frac{\partial v}{\partial x}$. If $\mathbf{x}$ is a vector, then we might get

$$\nabla_x f = \frac{\partial f}{\partial u}\nabla_v(u)J_x(v).$$

If these are all vectors, then

$$\nabla_x f = \nabla_u f J_v(u) J_x(v).$$

Now recall that $f_n = w^\top \phi_n + w_0$. Assume that $\phi_n = \sigma(W^1 x_n + w_0^1)$. We can write something like

$$\frac{\partial \mathcal{L}_n}{\partial W_1} = \frac{\partial \mathcal{L}_n}{\partial f_n} \nabla_{\phi_n} f_n J_{w_1}(\phi_n).$$

We now get

$$\phi_{nj} = \sigma(\Sigma_d W_{dj}^1 + W_{dj}^1).$$

We can now use the fact that the derivative of σ is $\sigma(1-\sigma)$.

So if we generalize, the first step is to compute the loss function with respect to the last layer, which can be written as

$$\mathcal{L}(\phi^L(\phi^{L-1}(\dots \phi(x))))$$

and store intermediate values. Now say we want

$$\frac{\partial \mathcal{L}_n}{\partial W^\ell} = \nabla_{\phi^L}(\mathcal{L}_n)\mathcal{L}_n \cdot J_{\phi^\ell}[\phi^{\ell+1}]$$

This is called reverse mode differentiation. You multiply things backwards, and you find derivatives. Reverse-mode differentiation is faster when the variables are all

§10 March 4th, 2019

Another paper did high-throughput physical phenotyping of cell differentiation.

So far, we've been covering models. We can think of a few objectives. There are least squares, L_0, hinge, and log-likelihood.

Maximum margin is a different type of *objective function*, not necessarily a different type of model. The intuition here is that it's "safer" for the point to be farther from the line than closer. One issue you have to deal with here is the presence of a point.

The boundary is $\mathbf{w}^\top x + w_0$. Let $x_n - x_p$ be parallel to w. Then the margin r looks like

$$x_n = x_p + r\frac{w}{\|w\|}.$$

The margin of a set of point is that smallest margin. The goal is now to maximize the margin of all the points. Alternatively, we can write our task as finding the minimal $\|\mathbf{w}\|$ such that

$$y_n(w^\top x + w_0) \ge 1.$$

This is a QP. This corresponds to finding

$$\min w^\top w \text{s.t.} y_n(w^\top x_n + w_0) \ge 1.$$

There's a tension between margin size as well as making mistakes.

The way to relax it is a soft-margin technique. So far, we've been doing hard-margin techniques. Given a line, you have a positive margin (≥ 1). We get that a good definition for an error term is

$$\xi_n = \begin{cases} 0 & \text{if outside margin} \\ |y_n - (w^\top x + w_0)| & \text{otherwise} \end{cases}$$

Then the objective will look like

$$\min \|w\|^2 + C\sum \xi_n.$$

The good news is that the function is still convex. The bad news is that we have to cross-validate to calculate C.

Now let's review some types of losses. A 0/1 loss is 0 on the correct side of the boundary and 1 on the bad side. The hinge loss is equal to the error. The log loss tries to push things out.

§11 March 6th, 2019

Recall that the margin of x_n is equal to

$$\frac{y_n(w^\top x_n + w_0)}{\|w\|}.$$

Hard margin-maximization is given by

$$\min_{w_1, w_0} \|w\|^2$$

such that $y_n(w^\top x_n + w_0) \geq 1$. Meanwhle, soft-margin classifcaation is simply

$$\min_{w_1, w_0, \xi} ||w||^2 + c \sum \xi_n$$

such that $y_n(w^\top x_n + w_0) \geq 1 - \xi_n$, where $\xi_n \geq 0$.

Gradient-descent on high dimensional data can be slow. Are there alternate ways of solving?

We can now find

$$\min_{w, w_0} \max_{\alpha} \frac{1}{2}\|w\|_2^2 - \sum_n \alpha_n[y_n(w^\top x_n + w_0) - 1].$$

If the constraints are violated, then α_n drives the objective to ∞. If the constraint is met, then α_n goes to zero. Lastly, if the constraint met with equality. We can now apply Von Neumann's minimax theorem, showing that this is equal to

$$\max_{\alpha} \min_{w, w_0} \frac{1}{2}\|w\|_2^2 - \sum_n \alpha_n[y_n(w^\top x_n + w_0) - 1]$$

Now, taking a derivative with respect to w we get $w - \sum_n \alpha_n y_n x_n = 0$.

We can now substitute it back in, and we get

$$\frac{1}{2}\left(\sum_n \alpha_n y_n x_n\right)^\top \left(\sum_{n'} \alpha_{n'} y_{n'} x_{n'}\right) - \sum_n \alpha_n y_n \left[\sum_{n'} \alpha_{n'} y_{n'} x_{n'}^\top\right] x_n - \sum_n \alpha_n y_n w_0 + \sum_n \alpha_n$$

Combining these gives us the expression

$$-\frac{1}{2}\sum_{n,n'} \alpha_n \alpha_{n'} y_n y_{n'} x_n^\top x_{n'} + \sum_n \alpha_n$$

We want to maximize this expression such that $\alpha_n \geq 0$, and $\sum_n \alpha_n y_n = 0$.

We can now write $w_* = \sum_n \alpha_n^* y_n x_n$. The prediction that we make is simply going to be the sign of

$$\sum_n (\alpha_n^* y_n x_n^\top) \cdot x + w_0.$$

The only points which get to vote is the α_n points which are on the margin, since other points have $\alpha_n = 0$. To solve for w_0, we can just find n such that $\alpha_n^* > 0$. In the objective, also, we only see x_n in products. As we know, sometimes x alone is not sufficient, and we need $x \mapsto \phi(x)$, in which case we sub in $\phi(x_n)^\top \phi(x_n')$. We can let $K(x_n, x_n') = \phi^\top(x_n)\phi(x_{n'})$ for some ϕ. Because the only part in the predictor which involves the x is $x_n^\top \cdot x$, we can replace it with something like $K(x_n, x_n')$.

In the original, we can write that we're trying to minimize

$$-\frac{1}{2}\sum_{n,n'} \alpha_n \alpha_{n'} y_n y_{n'} K(x_n, x_{n'}) + \sum_n \alpha_n.$$

The most important part here is that we don't need to specify ϕ!! It can even be infinite dimensional. That's how you can get the radial basis kernel,

$$K(x, x') = \exp\left(-\frac{\|x - x'\|^2}{\lambda}\right)$$

We now have the following result.

Theorem 11.1

For all f, if we have that

$$\int_x \int_{x'} f(x) f(x') K(x, x') \geq 0$$

then there exists an ϕ such that $K(x, x') = \phi(x)\phi(x')^\top$.

§12 March 11th, 2019

Phoebe Robinson and Jessica Williams come to ask you for a loan to make a pilot to pitch a series to HBO. You have to make a prediction about whether they'll pay back their loan? What should we think about this?

Credit is allocated on the basis of information about us. Is there information that banks and other private companies *shouldn't* be allowed to use in making judgments about creditworthiness.

On one hand, more information, more certainty. On the other hand, removing information of certain types will reduce the implicit bias risk. Even though their might be a risk about past and current injustice. Certain information that might yield an accurate prediction are in fact impermissible to appeal to because that connection in itself.

The motivation for these responses is a conception of the social good. This will include the principles and values by which social institutions establish basic rights and liberties, distribute scarce resources, and organize work.

Machine learning can be used to generate credit scores, predicting recidivism in prospective parolees, evaluating job candidates, and evaluating and diagnosing patients.

How does machine learning help us to realize socially good outcomes? What does it mean to do this correctly, and what does it mean to this ethically.

§12.1 Discrimination

Discrimination is a combination of disparate treatment and disparate impact. Disparate treatment involves classifying someone in an impermissible way. Some types of classification are morally neutral, whereas other types are morally problematic. This involves the intent to discriminate, and is evidenced by explicit reference to group membership.

Now we think about disparate impact, which looks at the consequences of classification / decision making on certain groups. No intent is required, and it is facially neutral. Practices with a disproportionate impact on a particular group do not ause disparate impact if they are "grounded in sound business considerations."

Sometimes Machine Learning algorithms discriminate, like in facial recognition.

§12.2 Discrimination as a Lack of Accuracy

Motivating idea: discriminating bias makes classification *less accurate*. We remedy this discrimination by building more accurate models. In hiring, machine learninng algorithms are free of human biases.

§12.3 Machine Bias

What if stuff you trained on was messed up? Sometimes the biases of humans are not mitigate by the machine learning algorithm. A machine learning system may be trained on *data infused with human bias*. The Northpoint software are baesd on prior arrests, age of first police contact, parents' incarceration record. This information is shaped by biases in the world. Then there was an algorithm that generated analogies from the software's vectors, such as man is to woman as computer programmer is to homemaker.

Machine learning algorithms can perpetuate discrimination because they are trained on biased data.

§12.4 When Accuracy Creates Discrimination

What is the aim of online advertising? We need to get clicks. The dataset is being generated from actual user behavior - unbiased. Compare this to our previously biased data sets.

The goal has been to make accurate generalizations in order to do ethical machine learning. What does this case show us about accuracy and discrimination.

§12.5 Discrimination Beyond Accuracy

First, is the task to be optimized one that contirbutes to the social good? It can be the case that if all of my customers are racist, then maybe I do have to be racist.

If the performance task for advertisements isn't just to get clicks, what could it be?

If we have a performance task that achieves some social good and meets the criteria we mapped out in steps (1) - (3).

§13 March 13th, 2019, Yann LeCun, Facebook AI Research

Today there was a difficult midterm. Instead, there's a talk by Yann LeCun, on the power and limits of deep learning. This is the first mind-brain behavior lecture.

§13.1 Overview

Deep Learning involves engineering science and inventing new artifacts. There is the telescrope engine, electromagnet, airplane, fertilizer, radio... The methods include creation, intuition, tinkering, exploration, experimentation, happenstance. This is guided by theoretical, conceptual, intuitive understanding.

§13.2 Natural Science: discover, study and explain phenomena

Theory often follows invention. The theory of optics came after telescope. Thermodynamics was made to explain the steam engine. Aerodynamics was done far after the sailboat. Information theory came long after the teletype. Computer Science was a theoretical field after the invention of the calculator.

The inspiration of deep learning is the brain. McCulloch and Pitts found that networks of binary neurons can do logic. Hebb found Hebbian synaptic plasticity, Wiener made cybernetics, optimal filter, feedback, autopoiesis, auto-organization. Rosenblatt developed the perceptron. AI these days is mostly supervised learning. You train a machine by showing it example instead of programming. When the output is wrong, you can tweak the parameters of the machine. This does well to change speech into words, images to categories, portraits to name, photos to caption, etc.

The paradigm of pattern recognition is to hand-engineer feature extraction, and then use a trainable classifier like a perceptron to find the result. Back then, perceptrons were motorized potentiometers.

From 1969 to 1985 there was no progress. The way you train multilayer nets is to use the chain rule. The reason was that the neurons were wrong. They were obsessed with the idea of binary neurons. Binary neurons are easier to implement: No multiplication necessary! Binary neurons prevented people from thinking about gradient-based methods for multi-layer nets. Backpropogation is only possible with sigmoid neurons, because there were "fast" floating point calculators. Empiricism works, but empiricism is slow and expensive. Theory allows us to prune our empirical search space. For example, we know to not look like perpetual motion. Some theories allow us to predict phenomena.

The idea of deep learning is that you want to take the image into low-level features, then mid-level features, and so on. The idea of supervised machine learning is functional optimization, and we use stochastic gradient descent. It's like walking the mountains in a fog and following the direction of steepest descent to reach the village in the valley. But each sample is noisy, since

Then we use biology and neuroscience who studied the visual systems that people see simple and complex cells.

We have a filter bank and nonlinearity. People in bell labs began building hardware. And then he realized he could recognize many images without segmentation, so that they could recognize many images. AT one point his system, 50% of the percents were correct. 49% stocks were rejected, and 1% were errors. ConvNet was applied to large images. Heatmaps at multiple scales.

Then from 1996 to 2006, training a character recognizer took 2 weeks on a sun or SGI workstation. A very small ConvNet by today's standard (500,00 connections). Then data was scarce and NNs were data hungry. The only large datasets besides character and speech recognition. Interactive software tools had to be build from scratch. The build an NN simulator with a cusom Lisp interpreter / compiler. But SN, SN2, and Lush gave us superpowers. Tools shape research directions. A support vector machine gives you guarantees on how close you can get to any function, but you can't do image recognition. It turns out that generalization bounds are really week.

There was a Vapnik-Jackel bet.

Then from 2006-2012 there was a lunatic fringe and the deep learning conspiracy. Why are provable results so important? I'm looking for my quarter I dropped? Did you drop it here? No, I dropped it two blocks down the street! People do research only on things they can solve. Some of us kept working on Neural nets. Soon they relized they could do semantic segmentation with Convolutional neural networks. Someone managed to implement convolutional nets on GPUs. Convolutional Neural Networks keep doing better. The layers keep getting deeper. Back then, convolutional neural networks were rejected from CVPR. Now you can only get accepted at CVPR if you use convolutional neural netwroks. ResNet50 and ResNet 100 are used routinely in production.

Now the question is, why are CNNs so effective, natural data is compositional, so things are efficiently representable hierarchially. Mask R-CNN can produce bounding boxes of people. Recently there was an application of computer networks for Medical Image Analysis. They can now find the segentation femur from MR images. Partial supervision is an interesting idea. You compute 3.5 billion images from instagram, and you use this to predict the hashtags.

There has been a lot of excitement with Deep Reinforcement Learning. It works great ... for games and virtual environments. Gerry Tesauro trained a system to play backgammon to win. They used a neural net to evaluate positions. We now call this deep reinforcement learning. Around 1995, the RL community abandoned neural nets. RL works for games. RL requires too many trails. RL often doesn't really work in the real world. The issue is that pure RL requires many, many trials to learn a task. It takes 50 million frames to match human performance. The best method (combination) takes 18 million frames (83 hours). RL works in simple virtual world than real time on many machines in parallel. Anything you do in the real world can kill you. You can't run the real world faster than real time.

To get to "real" AI. It would be nice as reasoning, learning models of the world, and learning hierarchical representatinos of actions. What we're missing is understanding the world. There's also learning hierarchical representations of actions. Neural nets with dynamic, data-dependent structure. A program whose gradient is generated automatically. We need a "hippocampus" (a separate memory module). Neural turing machine, memory networks, associative memory.The latest incarnations are tell a story.

The idea is you want to do automatic differentiation.

How do humans and animals learn so quickly? Babies learn how the world works by observation. Babies learn largely by observation, with remarkably little interaction.

Self-supervised learning is the idea where you predict any part of the input from any other part. Predict there is a part of the input you don't know and predict that. Essentially, predict the occluded from the visible. The machine has to predict every other part of the function.

RL gives very weak reward. Supervised learning predicts a category or a few numbers for each input. Self-supervised learning is predicting frames in videos. We can get a lot of feedback. Pure reinforcement learning is the cherry, supervised learning is icing. Self-superivsed learning solves millions of bits per sample.

The blind spot is hard to detect, because our brain does things with it. The idea of self-supervised learning is to train and figure out what it will looks like. This is called a masked autoencoder.

Self-supervised learning works well for text. We randomly mask 15% of tokens, and then we can recover the input. Bidirectional Encoder representations from Transformers. But high-dimensional continuous signals can be fixed.

Maybe we can learn predictive models of the world. To plan ahead, we must simulate the world. We then have things that observe if we are happy or not. But training samples are merely representatives of a whole set of possible outputs. There's a manifold of outputs.

One way to do this is adversarial training. Now you have a discriminator predict the differences in the world. We then run the generator. Initially it's going to be horrible. Train a discriminator which gives bad values.

§13.3 Self-driving cars

Then you can run the forward model with these sequences. We can backpropagate gradient or cost to train a policy network. Iterate. There is not need for planning at run time.

§14 March 14th, 2019

How could machines learn like animals and humans? But the thing is that those networks are all feedforward, but in the actual brain there are feedbacks. Some people have tried to use fMRI data to create a network from the brain. One idea is that there has to be a bottleneck in the machine learning that you have a blind spot.

Energy-based Unsupervised Learning. Let's imagine that we're a little animal and we have two eyes, so the world is two dimensional. We have activations of our right eye and our left eye. The idea is you want to push down on the energy of desired outputs, and push up on everything else. The reason we use energy and not probability is cause probability doesn't have to be normalized!

It turns out that you can interpret a bunch of classical algorithms in the energy surface. The energy function is the distance. When classifying a spiral, PCA doesn't do that well since it tries to linearize it. On the other hand, Z constrained to 1-of-K code, we can now minimize a value.

There's around seven ways to shape the energy function. The first is to build the machine so that the volume of low energy stuff is constant. We can push down of the energy of data points, push up everywhere else. We can also push down on the energy of data points, push up on chosen locations.

We might have y_i's, and the goal is to maximize

$$\prod_i P_w(y^i).$$

Taking the negative logarithm, we get that this is

$$\sum_i -\log p_w(y^i).$$

Now you need to need to minimize something like $f_w(y^i) + \frac{1}{p}\log_y \int_y e^{-\beta F_w(y)}$. Differentiating with respect to parameters gets you very close. When we take the gradient, we need can't easily take it, so we're basically done. This is basicaly pushing down of the energy of data points, push up everywhere else. If we only take samples when taking the integral, we can just push up on chosen locations.

Now LeCun's favorite technique is to use a regularizer that limits the volume of space that has low energy. He uses sparse coding.

Now we do a "decoder with regularized latent variable" model. We get sparse coding, and then we use sparse modeling (Olshausen Field 1997). We can regularize the latent representation, with a LASSO regularizer. You need to be careful that the decoder doesn't blow up.

Now we can use a use an encoder-decoder vs latent variables model. A linear decoder is predictive sparse decomposition. Basically, we can decompose everything into a ton of strokes.

We can look at convolutional sparse coding, where we replace dot products with convolutions.

The number of things you can find is really great.

One way to organize is to say that everything lies on a 2-D topology, and groups are patches of neurons. You get things that look like pinwheels, and that's the features you get. If we don't share weights convolutionally, we get more pinwheels. Another way to drive unsupervised learning is train an autoencoder to penalize slow features. If you take two features from a video, but after pooling, you want the representation to be the same. This enforces shift invariance.

Yesterday, he said that reinforcement learning is really bad. to reach human performance. They have to very fast. There's another example and it's alphastar, or deepmind's starcraft II bot. It's three neural networks which are trained automatically. It produces a distribution on actions. It is trained on supervised mode on recorded games. Then, this is used to prime a reinforcement learning to play starcraft. The thing is it needs to play starcraft for 200 years of play!

There are a few forms a self-supervised learning. One is to predict any part of the input from any other part. The general idea is to pretend there is a part of the input you don'tknow and predict that. But really this is an autoencoder. You feed it in, and then reconstruct the video clip. To predict the future you can't do an inference efficiently. Now the nice thing is that y and z is a complete representation. In fact,

§15 March 25th, 2019

For this part of the semester, we will focus unsupervised learning. Unsupervised learning works this way: instead of pairs $\{(x, y)\}$, our goal is to produce a summary of the data $x_1, x_2, \ldots x_n$. What are some examples in which this might be helpful? One major category is compression. Suppose that you have an HD video of very high resolution.

Small representations are helpful for compression applications. Another is data exploration / understanding, which can be used for organization as well as for generating hypotheses. A relevant term here is "representation learning". "Representations" have become equivalent to the term unsupervised learning.

§15.1 How to Evaluate Unsupervised Learning

There's a question of how to evaluate? One reasonable form of evaluation of a summary is to minimize some notion of reconstruction error. For example, this means to minimize

$$\sum_n d(x_n, f(g(x_n))),$$

where g is an encoding of x_n and f is the decoding. This is a very concrete metric. Say your summary is a cluster, that is x_n is mapped to a cluster center μ_k, which is the encoding.

§15.2 Clustering

A very natural choice for loss is L_2 loss, given by $\|x_n - \mu_{z_n}\|^2$, where z_n represents x_n's cluster ID. Today we're going to do this in a non-probabilistic way. We will further assume that the number of clusters K is given. Let z_n be a vector of length K. The vector z_n has a one-hot encoding.

The goal of K-means is to find

$$\min_{\{z_n, \mu_k\}} \sum_n \sum_k z_{nk} \|x_n - \mu_k\|.$$

This problem is NP hard, but we can try to solve badly.

§15.3 K-means clustering

The solution is something called blocked coordinate ascent. The idea is that there are two sets of unknowns, the means and the assignment and the idea is that given one variable it is very easy to solve for another one. Intuitively, we update $\{z\}$ given μ, and then update μ given $\{z\}$.

For $1, \dots N$, we assign z_n to

$$\operatorname*{argmin}_k |x_n - \mu_k|$$

Note that this process cannot make things any worse; this is key.

Secondly, for $1, \dots K$, we can take

$$\mu_k = \frac{1}{N_k} \sum_n z_{nk} x_n$$

You iterate through this on and on until you obtain convergence. Let's discuss properties of this algorithm. Note that it has monotonic improvement. Another nice property is that it's easily parallelizable and thus is well suited for distributed computing. Choosing the value of K is tricky, but it gets messy.

§15.4 Hierarchical Agglomerative Clustering (HAC)

The idea is to build a tree called a dendrogram. You do a tree DP, where you cluster vertices up the tree, such that two vertices are clustered if they belong to the same subtree.

Everyone starts in their own cluster and then we merge the closest clusters. To do this, we need a concept of distance $d(x, x')$ between two data points x. We also want a concept of distance between clusters. One definition is just the smallest distance between all pairs of points in each cluster. Another one is the maximum distance between all pairs of points in the cluster. The ward linkage is the distance between the averages. Max HAC encourages compact clusters, but min encourages stringy clusters.

§16 March 27th, 2019

Today we will be covering probabilistic mixture models.

§16.1 HAC vs K-means

Here's a concept question. Suppose that we have data points $0, 3, 4$ and $7, 8$ in one dimension, and we want to group them into two clusters. $0, 3, 4$ yields a result of $(49 + 29)/3 = 26 + 0.5$. $0, 3$ and $4, 7, 8$ yields $9/2 + 26$ which is bigger, so K-means gives us $0, 3, 4$ and $7, 8$. HAC may return either $[0, 3, 4]$ and $[7, 8]$ or $[0]$ and $[3, 4, 7, 8]$. $[0]$ and $[3, 4, 7, 8]$ also stick. K-means is less robust than HAC.

Imagine that our data is of two forms, some of which is

$$x_1 = \underbrace{00 \dots 0}_{k \text{ zeroes}}$$

followed by random noise and some data is

$$x_2 = \underbrace{11 \dots 1}_{k \text{ zeroes}}$$

followed by random noise.

§16.2 Mixture Models

K-means is going to win here, but HAC is not going to get stuck. These previous clustering methods have been somewhat tricky. We will now create a generative model, where there's a hidden cluster label z_n, each of which generates a x_n. For example, z_n might be the location of the car (US vs Europe), and x_n might be the weight of the car. We believe each data point is some part of a distribution, and then they were all mixed and then placed together.

In the classification setting, you are given a z_n and you can build the classifier. Now what we're saying is that we don't have that label available. z_n is a categorical distribution with a vector $[\pi_1, \pi_2, \dots \pi_k]$. We can get that, say, $x_n = \mathcal{N}(\mu_{z_n}, \Sigma_{z_n})$. Here, we're looking at

$$p(z|x, \theta, \pi) \propto p(x|z, \theta)p(z|\pi).$$

We write down a "complete" data log-likelihood. This is equal to

$$p(\{x_n, z_n\}|\theta)$$

We want

$$\mathcal{L}(\theta) = \sum_n \log p(x_n, z_n|\theta, \pi) = \sum_n \log(p(x_n|z_n, \theta)) + \log p(z_n|\pi).$$

Now we let this be

$$\sum_n \sum_k (z_{nk} \log(p(x_n|z_n, \theta)) + z_{nk} \log \pi_k)$$

Notice that there's 2-unknowns, which are $\{z_{nk}\}$, and there are global unknowns, which are $\{\theta, \pi\}$. The key idea here is that there are two unknowns and we optimize it in a way such that we're holding one set constant, optimizing the others, and then holding the other set constant.

§16.3 EM-algorithm

Now we're going over expectation-maximization. Let q_{nk} be my distribution on z_n. Assuming that there are four clusters, assume that there are four parameters in z_n. You can imagine a case in which the probabilities are .75 and .25. Now we'll look at $\mathbb{E}_z[\mathcal{L}_n(\theta)] = \mathbb{E}_z[\sum_k z_{nk} \log p(x_n|\theta_k)]$. We get that this is

$$\sum_k q_{nk} \log p(x_n|\theta_k) + q_{nk} \log \pi_k$$

Notice or recall that q_{nk} is easy given θ, π, where the probability of z_n given $x_n, \theta, \pi) \propto p(x_n|z_n, \theta)p(z_n|\pi)$. In the EM-algorithm, this is the E step. Then for the M step, we get an expression for $\pi_k = \frac{\sum_n q_{nk}}{N}$. We then get formulas

$$\mu_k = \frac{\sum_n q_{nk} x_n}{\sum_n q_{nk}}, \Sigma_k = \frac{1}{\sum_n q_{nk}} \sum_n q_{nk}(x_n - \mu_k) \cdot (x_n - \mu_k)^\top$$

§17 April 4th, 2019

§17.1 Bayesian Networks

A Bayesian network is a bunch of things going into each other. The idea is that a few things affect each other. They are direct acyclic graphs. Each variable is a consequence of the ones before it. If we have a graph where A goes into B, C, D, B goes into C, C goes into E we have

$$p(A, B, C, D, E) = p(A)p(D|A)p(B|A)p(C|A, B)p(E|C)$$

The reason this is helpful is that conditional independence means we have fewer parameters to train. The important thing is that $p(B|D, A) = p(B|A)$, and now we don't have to model as more complicated distributions.

For example, imagine a model such that general cleverness makes you a good test taker, which affects your exam grade. However, general cleverness may also help you understand material, which helps you with your homework. Understanding material also helps you with your exam grade.

If good test taking and understanding material cause exam grade, the grade and material are not independent. However, being a good test taker is independent of understanding material given cleverness.

§17.2 d-separation

If there's a path from A to C to B, observing C causes A, B to be conditionally independent.

Assume that C causes both A and B. Observing C blocks A, B. Now if A and B go into C, A and B are independent, but observing C makes A and B not independent. A causal graph is not the same as a DAG. A bayes net structure implies a factorization of a distribution, but it may not imply *causation*.

The exam grade is not conditionally independent of the homework grade, so that exam grade explains homework grade. Now we look at understanding the material. If you build a Bayes net a different way, you might be modelling the joint distribution correctly, but you may not be modelling the causation correctly.

You can have undirected graphs,that give you things like

$$p(A, B, C, D, E) \propto \phi(A, B)\phi(A, C)\phi(C, D)\phi(B, E, D)$$

The last form are factor graphs. You are still in proportional land here, and there are problems with normalizations.

Now let's do interventions!

In causal inference, you can intervene on a variable, forcing it to equal something, which lets you cut all the links going in.

Assume that A goes into B, C, C goes into D,E, and B goes into D. Assume that all variables are continuous and one dimensional, and they have a linear relationship of the form

$p(C|A)$ has three parameters, $p(B|A)$ has three parameters, $p(D|B, C)$ has nine parameters, $p(E|C)$ has three parameters, which is 18 in total. The other case has $1+2+1+1 = 5$ parameters. I'm wrong. A has two real parameters (for the distribution), B has 3(2), C has 3(2), D has 3(2), and E has 9(2).

§18 April 10th, 2019

There's a predictor for homosexuality. It looks like it just classifies people with beards and makeup. In china, they did a predicting criminality and see the people who are likely to have some issue. The key indicators look like whether you are smiling. There's some problem that if more crimes happen in lower income neighborhoods, then there should be more police there. The problem with policing is that ground truth labels are hard to ascertain.

§18.1 Bayesian Networks

Assume that A goes into C, B goes into C, and C goes into D. We can then find that

$$p(D) = \sum_{A,B,C} p(A, B, C, D)$$

We can write the summation as

$$p(D) = \sum p(A)p(B)p(C|A, B)p(D)$$

Assume that genes and smoking cause cancer. Bad genes and smoking gives you 99% chance, each of genes and smoking individually give you 9/10.

What is $p(G = 1|C = 1)$?

$C = 1$ has cases $G = 0, S = 1$, $G = 1, S = 1$, $G = 1, S = 0$.

The middle one is $99/100 \cdot 1/2 \cdot 1/4$. The first one is $1/2 \cdot 1/4 \cdot 9/10$. The third one is $1/2 \cdot 3/4 \cdot 9/10$.

The first one is 90. The middle one is 99, the last one is 270. The answer ends up being 297. We get

$$\frac{270 + 99}{270 + 99 + 90} = \frac{369}{459} = 80.4\%$$

What is $p(G = 1|C = 1, S = 0)$?

The second one is guaranteed.

Again, computing

$$p(D) = \sum_{C} p(D|C) \left[\sum_{B} p(B) \left[\sum_{A} p(A)p(C|A, B) \right] \right]$$

After you marginalize A, then the resulting is just a function of B, C. This is a function of C and B only.

Or we can write something like

$$\sum_{A} p(A) \left[\sum_{B} p(B) \left[\sum_{C} p(C|A, B)p(D|C) \right] \right].$$

The thing is that this object is extremely large. THe computational cost of variable elimination depends on the number of variables in intermediate factors. First, a poly-tree ordering is easy. One starts at the leaves and work in. poly-tree means that the undirected graph is also a tree. Start at the leaves and move in. For a loopy graph this is harder. The general idea is to query Q, evidence E, and then marginalize.

Assume that A goes into B goes into C. When finding $p(B|A = 1)$, we can note that this is not an ancestor of Q and E. If a node is not a parent of either Q or E it's totally fine. The other approach is to find leaves and work inwards toward the query.

For general Bayesian networks, finding an ordering is hard.

Say that A goes into B goes into C which goes into E, F, G respectively. We can write

$$p(c|E, F, G) = \sum_{A,B} p(A, B, C|E, F, G) = p(A|E, F, G)p(B|A, E, F, G)p(C|A, B, E, F, G)$$

We can simplify a lot of these terms. The probabilities become

$$p(A|E)p(B|A, F)p(C|B, G)$$

§19 April 15th, 2019

§19.1 Hidden Markov Models

Today we will finish inference in Bayesian networks: Hidden Markov Models and Linear structures. There might be a bunch of state evolutions like S_1 going into S_2 and so on. These have a very long history. We will assume that the states are hidden, and that $\{x_i\}$ are observed.

We have a prior $p_0(s)$. The generative process looks like $T(s'|s)$ There is some probability of s_2 given s_1, s_3 given s_2, and so on.

Lastly, we need the probability of x_i given s_i. Example: S is binary, and represents $\{\text{dry}, \text{rainy}\}$. x is a humidity measure, and is continuous. Assume that your prior is $p_0(s) = [.5, .5]$. Then we will ask $T(s'|s)$. This is where the markov chain comes in. There are two possible beginnings. We now need $\Omega(x|s)$. We can see $\Omega(x|\text{dry}) = \text{Unif}[0, 20]$ and $\Omega(x|\text{rainy}) = \text{Unif}[15, 100]$. Assume that dry to dry is .5, rainy to rainy is .75.

§19.2 What do we want to infer?

One common task we can do on a hidden Markov model is to compute $p(s_t|x_1, \dots x_t)$. This is in contrast to the task called smoothing, which is $p(s_t|x_1, \dots x_T)$. Smoothing is a post-hoc application. Smoothing comes up in language applications, where you get the entire sentence. We can also find the marginal likelihood, which is to compute $p(x_1, x_2, \dots x_T)$. This might be useful in anomaly detection. There's also best path, which is to find

$$\operatorname*{argmax}_{s_1, \dots s_T} p(s_1, \dots s_T | x_1, \dots x_T)$$

There's also prediction, which is finding

$$p(x_{t+1}|x_1, \dots x_t).$$

The question is, how do I get p_0, T, Ω, and given p_0, T, Ω, how do I perform the tasks.

Suppose that we had $\{s_1', \dots s_T', s_1^2, \dots s_T^2, \dots\}$. If we have a sequence, we can look at all the pairs and count the number of times we transition from dry to rainy. By counting that, we can fit T. If the s's are available, this will help us train the parameters. Say that you have s_3, and we are computing $p(s_3|x_1, x_2, x_3) = \sum_{s_1, s_2} p(s_1, s_2, s_3|x_1, x_2, x_3)$. However, the naive approach grows exponentially in T, so this isn't great. Now let's look at what the joint actually looks like, so we get that the expression is proportional to

$$\sum_{s_1} \sum_{s_2} p(s_1)p(x_1|s_1)p(s_2|s_1)p(x_2|s_2)p(s_3|s_2)p(x_3|s_3)$$

This can now be written as

$$p(x_3|s_3) \sum_{s_2} p(s_3|s_2)p(x_2|s_2) \sum_{s_1} p(s_1)p(x_1|s_2)p(s_2|s_1)$$

There is now a probability of getting $p(x_2|s_2)p(s_2|x_1)$. this is proportional to $p(s_2|x_1, x_2)$. The key idea is that we started with $p(s_1|x_1)$, then we have $p(s_2|x_1, x_2)$, and finally we can find $p(s_3|x_1, x_2, x_3)$. And finally, we can use this to find $p(s_3|x_1, x_2, x_3)$. We can now find

$$\alpha_1 \equiv p_0(s_1)\Omega(x_1|s_1)$$

Now we can define $\alpha_t = [\sum_{s'} \alpha_{t-1}(s')$ We can now find

$$\alpha_t = \left[\sum_{s'} \alpha_{t-1}(s')T(s|s')\right] \Omega(x_t|s)$$

We just need to normalize α_t. Each α does not sum to one, so we need to sum them. We want to find $p(x_1 \dots x_t|s_t)$. Now let's consider the backward variables: $\beta_T(s)$ is a vector of all 1s. β is the variable at the end. Now,

$$\beta_t(s) = \sum_{s'} \beta_{t+1}(s')T(s'|s)\Omega(x_{t+1}|s')$$

The point is that β tells you information coming from the back, so we can write that

$$p(s_t|x_1, x \dots x_T) \propto \alpha_t \odot \beta_t$$

We can find that $p(s_1, p_0, 0) = [0, 1, 0]$. Now there is probability 1.4 you are in A, 1/2 you are in B, 1/4 you are in C. Then we get $[0, 1/2, 1/2]$.

§20 April 17th, 2019

Today we start reinforcement learning.

§20.1 Pseudo-Labelling

One way to use the unsupervised is to provide labels on data you don't know about, but then cross-validate on data we do know about for sure.

§20.2 Planning/Reinforcement Learning

Assume that *our models are perfect*. Suppose that we have an HMM. In the Markov state sequence, actions combined with states determine r.

The model is that an agent sends actions to the world, and the world sends back some observations and some rewards. Goal: find a policy to maximize long-term expected rewards. However, this is hard. Instead, we will have the world send s, r back to the agent, as opposed to a weak indicator. We have that $\pi(s, a)$ is the probability of doing a in state s. The only policy is what to do in state s.

In the finite horizon, we are trying to optimize

$$\mathbb{E}\left[\sum_{\tau=0}^{T} r(s_t, a_t)\right]$$

In the infinite horizon, it might seem meaningful to optimize

$$\mathbb{E}\left[\sum_{\tau=0}^{\infty} r(s_t, a_t)\right]$$

but there might be many policies such that this is ∞. To combat this, instead optimize

$$\mathbb{E}\left[\sum_{\tau=0}^{\infty} \delta^\tau r(s_t, a_t)\right]$$

At the moment, assume your MDP has a set of states s. We let there be a set of actions a. We also have $T(s'|s,a)$, an $R(s,a)$, an a δ.

There's an interesting data. In the finite horizon case, the policy depends on time. There's no reason to invest in something tomorrow.

Simpler question: how can we evaluate a policy π? The value of a policy at s is

$$V^\pi(s) = \mathbb{E}\left[\sum \gamma^t r_t | s_0 = s\right]$$

Then we write

$$V^\pi(s) = \mathbb{E}\left[r(s,\pi(s))\right] + \gamma \mathbb{E}[V^\pi(s_1)]$$

because the problem is inherently recursive. We end up getting something like

$$V^\pi(s) = r(s,\pi(s)) + \gamma \sum_{s_1} T(s'|s_1,\pi(s)) \mathbb{E}\left[\sum_{t=0} \gamma^t r_t\right]$$

This becomes the **Bellman Equation**, which is

$$V^\pi(s) = r(s,\pi(s)) + \delta \sum_{s'} T(s'|s,\pi(s)) V^\pi(s')$$

The way you solve an equation like this is by repeatedly applying updates, letting

$$V^\pi_{k+1}(s) \leftarrow r(s,\pi(s)) + \gamma \sum_{s'} T(s'|s,\pi(s)) V^\pi_k(s').$$

We're trying to find a fixed point, which will represent the correct values.

They way to interpret this is that the future is being increasingly discounted, and the things you collect along the way is the overall reward.

To find the optimal policy, one method is **policy iteration**. We can define

$$Q^\pi(s,a) = r(s,a) + \delta \sum_{s'} T(s'|s,a) V\pi(s')$$

The idea is to compute V^π and Q^π. We change our policy by making one deviation from the current policy. In the improvement step, we change π to be

$$\arg\max_a Q^\pi(s,a)$$

Like Cinderella, assume that one different step changes your life forever and makes it better.

§21 April 22nd, 2019

Specifying the reward function is difficult. Let's look at a grid world. We have a start position and we have an end position.

Let's say that we start at a policy π. We will evaluate $Q^\pi(s,a)$. Then we take π' to be greedy with respect to $Q^\pi(s,a)$.

The value function of being in state 1 is 9. The value function of being in state 2 is 8. The value of state 3 is 4.

Now we find the Q function. This is a table with actions and states. The Q table tells you can do better by going on an excursion. The important point here is you get the reward of the point you start on. Keep this heuristic. The advantage is that policy iteration usually converges to the optimal policy.

§21.1 Value Iteration

Note that the optimal policy π satisfies:

$$V^*(s) = \max_a \left[R(s,a) + \delta \sum_{s'} T(s'|s,a) V^*(s') \right]$$

In this case, the approach is to set V_0 to *any* V, we can now set V_k to be the maximum over a of the policies from each thing. This will converge to the optimal set of values.

In planning, you have T, R and you get π^*. But in reinforcement learning, you have a simulator and are trying to find a π^*.

The idea is that you can explore and exploit a trade off. You can only learn from experience. You can update online / evaluate cumulative rewards. These are things you have to deal with when you are a simulator.

You have data s, a, r and so on, and you can try to learn what T, R are and solve it with planning.

The other way is directly policy optimization, where we want to find

$$\pi_\theta(s) = \theta_1 v + \theta_2 d + \theta_3 (v \cdot d)$$

Then, we can find

$$\frac{dV(s)}{d\pi_\theta(s)}$$

This requires a lot of data, because this is the *farthest away* from the data.

The last class of these approaches is called a value function approach. The idea is to learn $V(s)$ online. We obtain

$$V^*(S) = \max_a Q^*(s,a) = \max_a R(s,a) + \delta \sum_{s'} T(s'|s,a) V^*(s')$$

There, we see that $Q^*(s,a) = R(s,a)$ added to the value of being in some future state.

Then, the function $Q(s_t, a_t)$ should be set equal to

$$Q(s_t, a_t) + \alpha_t \cdot [r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

§22 April 24th, 2019

In playing go, the key insight in playing chess was about tree search.

We're now going to go back to discrete states.

Let

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_t (r_t + \gamma Q(s_t, a_{t+1}) - Q(s_t, a_t))$$

In order to get convergence, we need to try all (s, a) pairs.

The exploration strategy is ϵ-greedy: take the best action, but with probability ϵ take a large number.

This algorithm is on-policy.

The other type of rule is Q-learning, where we use

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_t (r_t + \left(\gamma \max_a Q(s_{t+1}, a)\right) - Q(s_t, a_t))$$

This is off-policy, because we didn't do a.

If we had a grid world, where there's a cliff, when there is uncertainty, SARSA and Q-learning will each

When action spaces are continuous, we want to approximate the function. Fitted Q iteration is

$$Q^k(s,a)r(s,a) + \max_{a'} \gamma Q^{k-1}(s', a')$$

It is good to be able to fit the reward function.Fitted Q-iteration lets you fit functions like Q^k. Because it gfits regressos one at a time, it is highly efficient.

Thompson sampling is another idea.

§23 April 29th, 2019

The midterm is on Wednesday, and the midterm is in the Geological museum. Hints as to what will appear on the exam: coordinate ascent (EM, Lloyd) and basic matrix derivatives (have fluency with matrix derivative).

What we've learned: we can now understanding problem types, and avoiding type mismatch. We also now understand solution process (and how to debug).

We have also learned to understanding solution processes and how to debug them. There may be situations where the formal objective may be $0-1$ loss, but we optimize hinge loss.

Interpretability provides a human-understandable explanation for the model's behavior. For example, "for patient X, BP was key". While you can't prove things, you might be able to use proofs to show things.

It might be the case that specification is difficult. For example, we want to tell a car what safety is, but there's a lot of gradations to that. It is hard to apply laws to AIs, because if the AI is not interpretable, we can't figure out if it's breaking the law.

There are visualization methods, where we visualize the data. One can make models intepretable from the beginning (white-box methods).

You can also force the model to be a generalized linear model, or an additive model, or so on. There's also, I recommend treatment X because it worked for other patients like you... Lastly, we have sparsity-based models which force the models to be small.

We can also do post-hoc interrogation of models. One can do sensitivity analysis, and look back and forth. One can also train a mimic model to imitate the black box. Does interpretable machine learning improve the number of journal publications?

In CS 281, there are fewer models but more focus on inference. CS 282 is reinforcement learning. CS 287 is natural language processing.